Towards UML Profile for Human Machine Interface Applications of In-vehicle Infotainment Platforms

Hemant Sharma, Dr. Roger Kuvedu-Libla, and Dr. A. K. Ramani

Abstract—**UML Profiles provided automotive software designers a way to customize the UML to their particular domain and purpose. In this paper, we fulfill the demand for a suitable profile for In-vehicle Infotainment and Telematics domain by sketching a UML profile for Human Machine Interface applications. The HIT-profile classifies different application and framework components, and enables their mapping to framework APIs. The profile concentrates on the structure of HMI application and framework interfaces, and utilizes UML 2.1 standard for the behavioral modeling.**

***Index Terms*— UML, Infotainment Human Machine Interface Framework (iHMIFw), Model Driven Architecture (MDA), HIT-Profile.**

I. Introduction

The development of modern In-vehicle Infotainment and Telematics systems requires reducing the gap between traditional hardware and software designs. Unified Modeling Language (UML) [1] is widely used in software development, and now it is heavily used in automotive software system design. Especially the latest release of the language, UML 2.1[11] with its extension proposals, brings several advanced features to support also this domain. UML 2.1 holds a promise of a general design language that can be understood by system designers as well as software and hardware engineers.

Recent trends in automotive software design indicate that the use of prefabricated building blocks for software development is on the rise. The prefabricated artifacts are the off-the-shelf (COTS) software infrastructure and domain-specific service components that one can acquire from different vendors and integrate them to deploy large-scale software applications. Vehicle Navigation is a good example of such an application on In-vehicle Infotainment systems.

Hemant Sharma is Software Engineer at Delphi Delco Electronics Europe GmbH, Bad Salzdetfurth, Germany. (e-mail: hemant.sharma @ delphi.com).
Dr. Roger Kuvedu-Libla is EMC-Comp.-Leader at Delphi Delco Electronics Europe GmbH, Bad Salzdetfurth, Germany.
(e-mail: roger.kuvedu.libla @ delphi.com).
Dr. A. K. Ramani, is Professor at School of Computer Science, Devi Ahilya University, Indore, INDIA. (e-mail: ramani.scs@dauniv.ac.in).

Creating models of user interfaces is not a completely new idea. In recent years, research has emphasized modeling the interaction and navigational part of user interfaces [1, 2]. The UML has grown into the most widely used modeling language, and it is used by many processes to specify models of the software system under development. It is already capable of modeling some of the aspects of HMIs. First of all, there is the use case diagram to capture user requirements before creating the user interface. From these requirements, designers can make conclusions for the HMI, and the HMIs can later be tested against them. The UML can flex its muscles when it comes to modeling the details of every interaction with the system. Designers can use the semantic HMI model to create a sound system and give exact instructions to the developers who will do the lexical design of the system and implement it. But just like the processes, the UML does not provide a means to model all the aspects of the HMIs.

Generation of HMI using XML based tools, currently becoming topic of interest, is inspired in Model Driven Development (MDD) paradigm and in Model Driven Architecture (MDA) [10]. This approach consists of designing UML-based models for HMI, from which generative code techniques are applied to automatically produce HMI code and other software artifacts. MDD paradigm has the goal to describe the system functionalities using a set of models, shifting the software development focus from code to models artifacts. An HMI model is a representation of how the end-users interact with the software system. MDD paradigm intends to create automatic mechanisms to generate software artefacts from these models.

This paper presents a new UML 2.1 profile, called "*Profile for HMI of Infotainment and Telematics System (HIT-Profile).* HIT-Profile defines a set of stereotypes for extending UML meta classes as well as design practices to describe HMI applications, target software platforms, and mapping of them. It is especially targeted to In-vehicle Infotainment and Telematics HMI framework implementation using UML 2.1 description. For this reason, HIT-Profile is used with a set of tools as depicted in Figure 1. These tools include the UML tools Artisan [14], Telelogic Rhapsody [15] and Enterprise Architect [16].

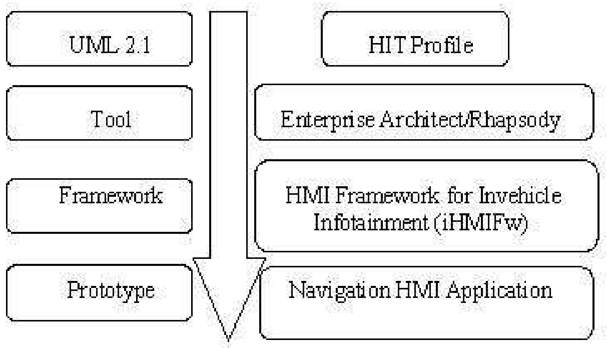

UML 2.1
HIT Profile
Tool
Enterprise Architect/Rhapsody
Framework
HMI Framework for Invehicle Infotainment (iHMIFw)
Prototype
Navigation HMI Application

Figure 1: HMI Design Flow using HIT-Profile.

Rest of the paper is organized as follows: In the following section an overview of related research is provided. Section 3 shortly explains the artifacts of Infotainment HMI. In section 4, we describe the HIT-Profile construction. Section 5 describes an example for model driven HMI development using HIT-profile. In section 6, we elaborate the future activities and finally conclude the paper.

II. Background

A number of extension proposals have been presented for real-time and embedded system design. The proposals can be roughly divided into three categories: system and platform design, performance modeling, and behavioral design. The Embedded UML [3] is a UML profile proposal suitable for embedded real-time system specification, design, and verification. A UML Platform profile is proposed in [1], which presents a graphical language for the specification. It includes domain-specific classifiers and relationships to model the structure and behavior of embedded systems. The ACCORD/UML profile [6] defines a methodology for model mappings during the different development stages. The UML-RT profile [5] defines execution semantics to capture behavior for simulation and synthesis. The UML Profile for Schedulability, Performance and Time (or the Real-time UML Profile) is standardized by OMG [4]. The profile defines notations for building models of real-time systems with relevant Quality of Service (QoS) parameters.

The UMLi [7] approach proposes a profile to capture the conceptual, presentation and behaviour aspects of systems. The UWE [8] approach focuses particularly on modeling Web systems. The proposals in UMLi and UWE, for the presentation design, have some similarities with our HIT-Profile Interaction View.

Within the scope of MDA, for a given language (or a simple metadata collection) two ways exist to define the language in terms of the actual OO meta-modeling techniques: (a) by defining a UML profile; (b) by defining a completely new metamodel through MOF. UML profile is a specific way of using UML to define (by means of a set of UML extension mechanisms like stereotype, constraint, tagged value, etc.) a specialized UML metamodel[12,13]. The second approach, instead, is a more desirable solution when clearly UML cannot capture nuances of design paradigms.

XIS Profile [9] promotes a platform-independent design for interactive systems. This means that the XIS profile allows the design of interactive systems at a PIM level ("Platform-Independent model", according to the MDA terminology [10]), so systems can be targeted, using specific model-to-code transformations, to different source-code languages and platforms, such as Web, desktop or mobile platforms (e.g., J2ME, .NET Compact Framework, or any embedded systems that are meant to support interactive systems).

III. Modeling Infotainment HMI Artifacts

A. *HIT-Profile Approach*

HIT-Profile extensions are used to define the structure and parameters of components of an HMI application and HMI framework platform as well as their mapping. Correspondingly, the HMI application design is divided into three parts: application description, framework description, and mapping. Both the application and framework descriptions can be developed independently of each other.

HIT-Profile mainly concerns the structure of an HMI application and our In-vehicle Infotainment HMI Framework (iHMIFw). The application is seen as a set of active classes with an internal behavior. The framework is seen as a component library with a parameterized presentation in UML for each library component. The profile does not restrict the behavioral modeling, and by default, it utilizes standard UML concepts for this.

HIT-Profile classifies different application and platform components by defining various stereotypes and strict rules how to use them. The objective is to enhance the support of external tools [14, 15, 16] for automatic analyzing, profiling, and modifying the UML description of an infotainment HMI system. The classification also assigns defined parameters to proper components.

In practice, the profile is applied in a tool framework depicted in Figure 1. The platform mapping can be explicitly performed by the designer, or assisted with tools. For the profile, we plan to develop a UML profiling tool that shall combine the UML description (model parsing) and simulation statistics (simulation log-file) that is obtained during the verification phase. Based on the profiling, also the HMI description will be modified to fulfil context related constraints.

B. *Infotainment HMI Framework Architecture*

This section provides an overview of architecture of our HMI framework, for which HIT-profile is expected to serve as description language.

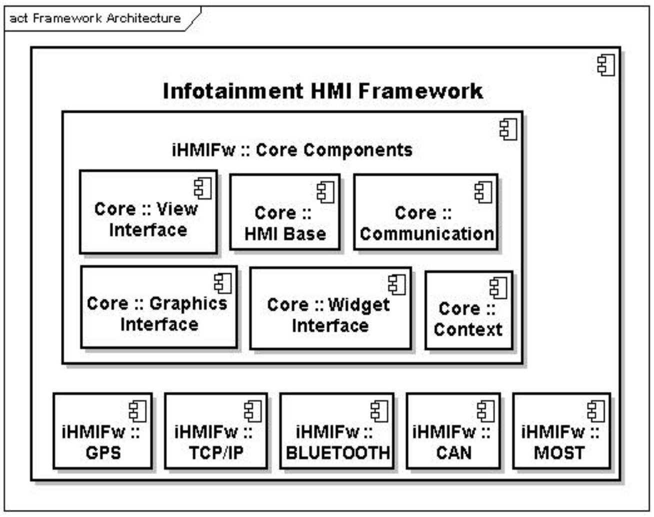


Figure 2: iHMIFw Components.

The framework has been developed using MDA methodology and partitioned into independent components based on functional area. Figure 2 partially represent the components of the HMI framework. The components are organized to make the framework scalable and flexible. The framework has a set of core components and some optional components. Core components provide the bare minimum functionality that is required for an HMI application. Optional component may be configured along with framework to provide additional functional interfaces.

View component is responsible for HMI View creation. It provides interfaces to describe the appearance and behavior of the view. Further, this component is also responsible to define the view tree structure for an HMI application and defining the view state transition. *HMIBase* defines the structure of HMI applications. This component enables the HMI application to interface with platform specific aspects such as startup and shutdown of application. *Communication* is responsible for communication between components of HMI Application as well as communication with external applications. *Graphics Interface* component interfaces to graphics resources in the system. This includes graphics libraries, display access handlers, image management and font resource files. HMI Applications are allowed to use specific widgets available in infotainment application specific widget libraries. *Widget Interface* component provide interface to the widget resources, the HMI application intend to use. *Context* component provides interfaces that help HMI Applications to define the context for its individual views. Further, this component has mechanism to read context configuration information from a XML file.

The components of iHMIFw shall be included into the models of HMI Application using HIT-Profile constructs.

C. *Metamodel for Infotainment HMI Applications*

The metamodel for HMI applications of In-vehicle infotainment system is shown in Figure 3.

The HMI applications shall inherit the layered component structure from iHMIFw. Presentation layer of the application will be composed of a set of hmiView meta classes. Every view must have an appearance and an associated behavior. Therefore, the interface classes *hmiViewAppearance* and *hmiViewBehavior* will be realized by applications. Depending on the features implemented by a view, it will have multiple widgets, realized using *hmiWidget*, contained in one or more containers, realized using *hmiWidgetContainer*. Actions and resulting response on presentation layer will be managed by instances of *hmiController* and its extensions. The applications will have the ability to retain the knowledge about their context by realization of *hmiContext* extensions.

Data and control communication between application and framework components as well as the external communication is managed by instances of communication and service interfaces i.e. *hmiCommunication* and *hmiServiceInterface*.

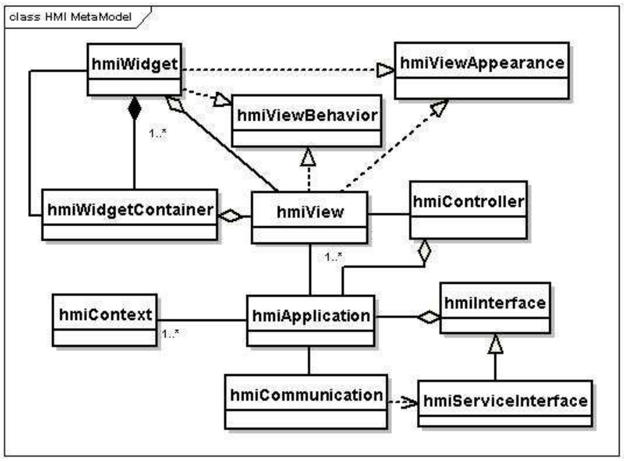


Figure 3: HMI Application Metamodel

IV. HIT Profile Construction

A. *HIT –Profile Stereotypes*

HIT-Profile contains stereotypes to support infotainment HMI design. The structure of the HIT-Profile is presented in Figure 3. Stereotypes of the profile have been partitioned into separate packages, namely, application stereotype package and framework stereotype package.

Application stereotypes help in modelling the HMI application structure whereas framework stereotypes map the iHMIFw constructs into application.

A view is composed of appearance and behaviour. An application is composed of application components, which are instantiated as application processes. Next, application processes are grouped into process groups.

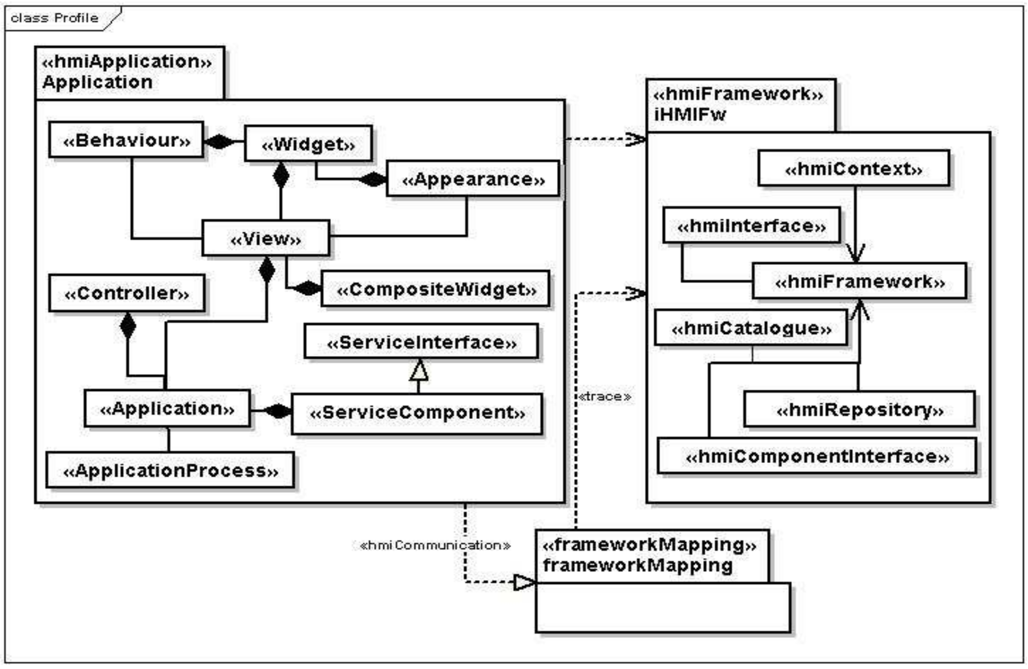


Figure 3: HIT-Profile

Stereotype Name (extended metaclass)	***Description***
Application(class)	Top Level HMI Application class.
ApplicationProcess (Structural Feature)	Instance of functional HMI application component.
Behaviour (class)	Behavior description of HMI component.
Appearance(class)	Static Design Description of HMI components.
Controller (Structural Feature)	Work flow description of HMI.
CompositeWidget (Structural Feature)	Collection of Basic Widgets
ServiceInterface (class)	Generic Interface for Internal and External Communication.
ServiceComponent (Structural Feature)	Collection of Feature Implementation.
View (class)	Static Definition of HMI Screen.
hmiContext (class)	Scenario Description class.
hmiCatalogue (Structural Feature)	Data and Function Repository.
hmiFramework (Structural Feature)	Top Level HMI Framework class.
hmiInterface (class)	Framework Interface for HMI Application.
hmiRepository (Structural Feature)	Persistent Storage for HMI.
hmiCompInterface (Structural Feature)	Communication Description Interface.

Table 1: HIT-Profile Stereotypes summery.

Correspondingly, iHMIFw is composed of components, which are instantiated as component instances. Finally, process groups are mapped to framework component instances using framework mapping. The summary of stereotypes of HIT-profile is presented in Table 1.
The stereotype <<Application>> is applied to define the top-level class of an HMI application description. The top level class has a class hierarchy defining different classes of an application. The active classes having behaviour are called functional components. Passive classes are structural components, which do not have behaviour, but instead, define composite structures and data structures storing application data. Process grouping, represented by <<ApplicationProcess>>, is a part of an HMI application description and it defines the process structure for HMI application. The structure is implementation-oriented and may thus differ from the composite structure of an application. Stereotypes <<Behaviour>> and <<Appearance>> describe an HMI view. Logical organization of all widgets of an HMI application is represented by stereotype <<View>>. Interface to framework components and other applications in the system is described by <<ServiceInterface>> and <<ServiceComponent>> stereotypes. <CompositeWidget>> represents a collection of widgets that belong to a view.

B. Tagged Values for HIT-Profile Stereotypes

The definition of HIT-Profile stereotypes is associated with tag definitions. These tags of stereotypes shall map to either attribute or interface of iHMIFw. Table 2 lists the tagged values for some of the structural stereotypes of HIT-Profile.

Tagged Values	*Description*
<<Application>>	
ID	System-wide unique identifier
Server	Application for which HMI is client.
Context	Supported context information.
<<Controller>>	
Controller States	List of Controller States
Current State	Current state of controller
<<ServiceComponent>>	
ID	Service identifier.
Interface List	List of provided interfaces
State	Component Availability state
<<hmiFramework>>	
State	Current State of Framework
Config	Framework Configuration Information.
CommInterface	Framework's internal and external Communication Interface
<<hmiRepository>>	
Type	Type of repository (Data/ Function)
Size	Maximum size
<<hmiComponentInterface>>	
Subscription	Application component list
InterfaceList	List of exposed interfaces.

Table 2: Tagged values for HIT-Profile Stereotypes.

C. Mapping of Stereotype to iHMIFw Interfaces

When both an HMI application and corresponding customization of iHMIFw framework have been modelled, each group of stereotype is transformed to corresponding implementation feature of framework component.

HIT-Profile	*iHMIFw*
<<Application>>	*iHMIFW_clApplication* HMI Application class
<<Controller>>	*iHMIFw_clStateContext* State machine base class for HMI Application
<<ServiceComponent>>	*iHMIFw_clSUCompBase* Base class Service Component.
<<hmiContext>>	*iHMIFw_clContextBase* Context base class for HMI Application.

Table 3: HIT-Profile mapping.

In order to achieve seamless model transformation, we have defined mapping of HIT-profile concepts into the iHMIFw classes and class members. Since HIT-Profile is mainly a collection of structural notations for expressing HMI application's composite structure and is orthogonal to the functional behaviour, we do not consider generation of functional code from UML models, which has been covered adequately by many commercial UML tools [14, 15, 16]. Instead, we focus on the mapping of structure of HMI application, context description and behavioural constructs. Table 3 shows some example mappings.

V. HMI MODELING USING HIT-PROFILE

This section presents HIT-Profile utilization with the design of a custom HMI application feature for Infotainment. The examples show how HIT-profile has been applied along with UML 2.1 diagrams, and how the HIT-Profile can be applied in Model Driven design.

A. Example Scenario – HMI for Off-board Navigation

The HMI application is intended to show running guidance and provide turn-by-turn navigation instruction. This application shall be part of infotainment system which also hosts the Navigation server application. Navigation server application provides the guidance relevant information update to HMI application. The HMI application is based on iHMIFw and controls the guidance running at navigation server application.

B. Platform Independent modeling

The design of the navigation HMI application description will start from the definition of the class hierarchy. The top-level application class and its components are created, and the associations between components are defined.

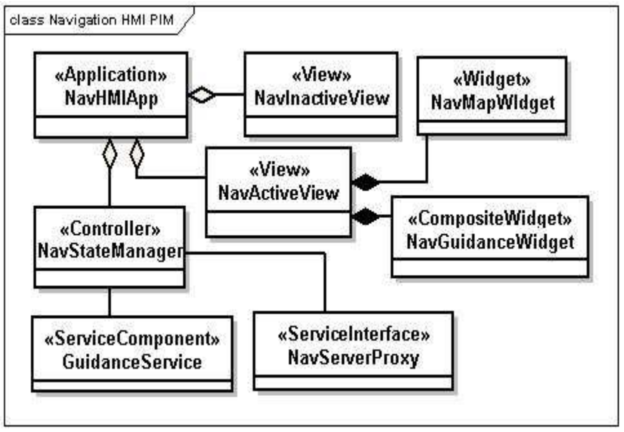

class Navigation HMI PIM
«Application» NavHMIApp
«View» NavInactiveView
«Widget» NavMapWidget
«View» NavActiveView
«Controller» NavStateManager
«CompositeWidget» NavGuidanceWidget
«ServiceComponent» GuidanceService
«ServiceInterface» NavServerProxy

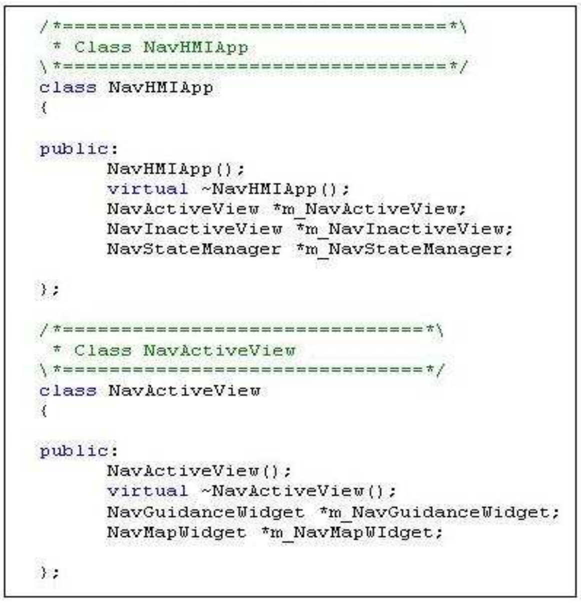
Figure 6: PIM for Navigation HMI Application

NavHMIApp is the top level class of the application. Thus, it is stereotyped as <<Application>>. The application contains *NavActiveView* and *NavInactiveView* as route guidance views and therefore stereotyped as <<View>>. *NavMapWidget* and *NavGuidanceWidget* are examples of widgets in a view and therefore stereotyped as <<Widget>> to compose the active navigation view. The application interacts with Navigation Server application via NavServerProxy Interface. Behaviour of views is controlled by <<Controller>> stereotyped NavStateManager class along with <<ServiceComponent>> stereotyped GuidanceService class.

C. Platform Specific Mapping

For the transformation of PIM to C++ PSM, we use the UML tool Enterprise Architect [16] with customized transformation script. The tool is able to interpret the models, and generate code out of them. The transformation script has been developed using template based approach. Description of details of transformation script is not the scope of this paper.

Figure 7 shows the source code of *NavHMIApp* and NavActiveView classes obtained from PSM transformation.



```
/*==================================*\
 * Class NavHMIApp
\*==================================*/
class NavHMIApp
{

public:
        NavHMIApp();
        virtual ~NavHMIApp();
        NavActiveView *m_NavActiveView;
        NavInactiveView *m_NavInactiveView;
        NavStateManager *m_NavStateManager;

};

/*===============================*\
 * Class NavActiveView
\*===============================*/
class NavActiveView
{

public:
        NavActiveView();
        virtual ~NavActiveView();
        NavGuidanceWidget *m_NavGuidanceWidget;
        NavMapWidget *m_NavMapWIdget;

};
```

Figure 7: Partial PSM for Navigation HMI Application

The example shows that HIT-profile can effectively describe the application and framework design aspects of an Infotainment HMI application. Mapping of stereotypes of the profile to framework APIs, visualize HIT-profile as Architectural Description Language (ASL) for iHMIFw.

VI. CONCLUSION

The creation of profiles and UML extensions is necessary when the standard model constructs do not give the expressiveness required to represent the specific characteristics of particular domains, as it happens with the HMI modeling process for In-vehicle Infotainment applications.

The UML profile for Infotainment HMI here presented, defines a language that allows specifying, analyzing, designing, constructing, visualizing and documenting the software artifacts in a HMI application design flow, providing a modeling framework for infotainment software systems in which high level functional models can be refined down to an implementation language.

There are, however, some open issues that need to be investigated further. In the first place, we think that platform-independent UML profiles are not enough, especially for Infotainment HMI application framework. Thus, iHMIFw framework is semi-developed HMI application that leverages the user to implement most of the common services, facilities and functionality of the applications. In this sense, platform- independent models are useful but insufficient for documenting application frameworks, since some details about the platform specific HMI implementation needs to be described, too. Another issue is about interoperability. We cannot make the simplistic assumption that all systems will be designed and modeled using the same modeling tools. Therefore, appropriate bridge between models is needed.

As part of our future work , the HIT-profile will further be enriched by the specialization of the stereotypes and by improvement of parameterization. It is our goal to get more experience to build customized HMI patterns, and flexible UML tool plug-in that allows simplifying the entire building process of infotainment HMI software artifacts.

REFERENCES

[1] D Silva, Paolo Pinheiro & Paton, Norman W. (2000): UMLi: The Unified Modeling Language for Interactive Applications. In: <<UML>> 2000 - The Unified Modeling Language: Advancing the Standard. LNCS Vol. 1939. Springer, pp. 117-132.
[2] Dolog, Peter; Bieliková, Mária (2002): Hypermedia Modeling Using UML. In: Hanacek, Petr: Proc. of ISM'2002, April 2002.
[3] G. Martin et al., "Embedded UML: a Merger of Real-Time UML and Co-design". Proceedings of the 9th International Symposium on Hardware/Software Codesign, April 2001, pp. 23-28.
[4] OMG, "UML Profile for Schedulability, Performance, and Time Specification". September 2003.
[5] B. Selic, and J. Rumbaugh, "Using UML for Modeling Complex Real-Time Systems, White Paper, Rational". March 1998.
[6] P. Tessier et al., "A Component-based Methodology for Embedded System Prototyping". Proceedings of the 14th IEEE International Workshop on Rapid Systems Prototyping, June 2003, pp. 9-15.
[7] Silva, P. P., and Paton, N., "User interface modelling in UMLi. IEEE Software", 20(4), July–August 2003.
[8] Hennicker, R. and Koch, N.,"Modeling the User Interface of Web App. with UML", Workshop of the pUML-Group at UML *2001*.
[9] Silva, A. , "The XIS Approach and Principles". Proceedings of the 29th Euromicro Conference, IEEE Computer Society, 2003.
[10] OMG. Model Driven Architecture. http://www.omg.org/mda/
[11] OMG, "UML 2.1.1 Specification". Feb. 2007.
[12] Vincenzo Grassi, Raffaela Mirandola and AntoninoSabetta, "A UML Profile to Model Mobile Systems". Seventh International Conference on UML Modeling Languages and Applications, UML 2004. Lisboa, Portugal.
[13] Alan Moore. Extending the UML RT profile to support the OSEK infrastructure. In Proceedings of Fifth IEEE International Symposium on OO Real-Time Distributed Computing, pp.341-347, 2002.
[14] Artisan Software website. http://www.artisansw.com .
[15] ILogix website. http://www.ilogix.com.
[16] Sparx Systems Website. http://www.sparxsystems.com.